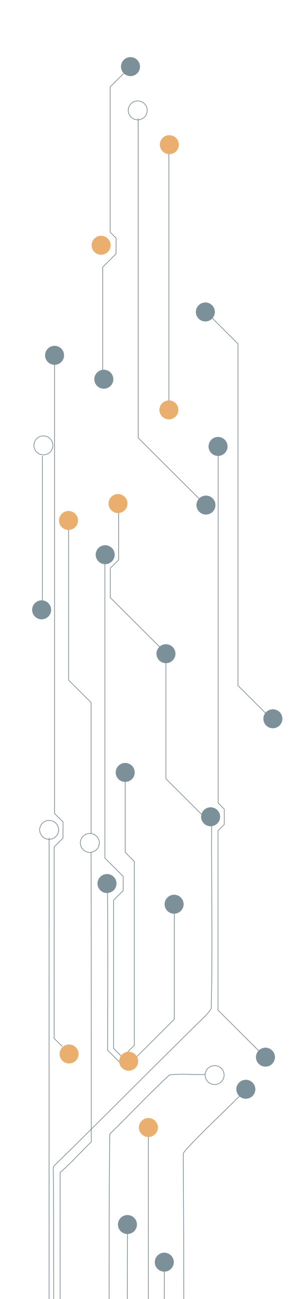
Programación estructurada



Índice



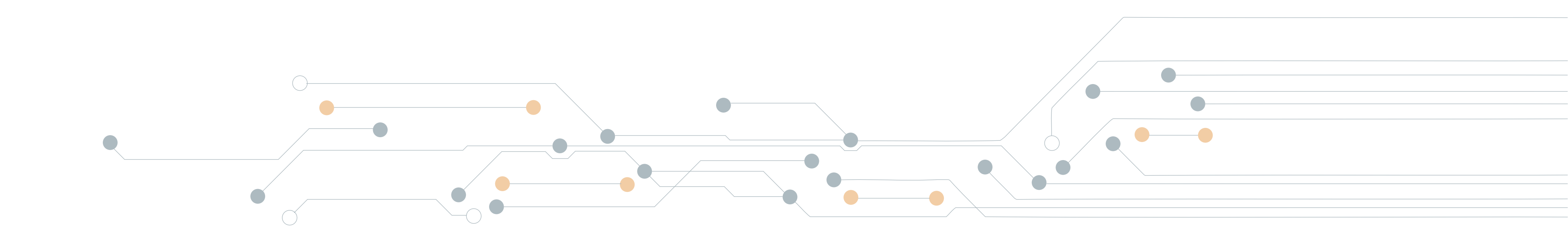
Programación estructurada

1. Programación estructurada

Como ya hemos visto, un algoritmo es un conjunto de operaciones que resuelven un problema específico. Pero muchos problemas de programación, especialmente los que tienen cierta complejidad, pueden resolverse con diferentes algoritmos, algunos de ellos más óptimos y eficientes que otros.

Esta situación se nos planteó durante la realización de los ejemplos de pseudocódigo que vimos anteriormente, cuando el programa que se encargaba de calcular la suma entre 1 y un número leído lo resolvimos de dos maneras, una utilizando saltos de un punto a otro del programa y la otra mediante el uso de una instrucción repetitiva, siendo esta última la más eficiente.

De ahí que a finales de los años sesenta surgiera una forma de programar que permitía crear programas fiables y eficientes, además de fáciles de comprender. Esta nueva forma de programar se definía a través del teorema del programa estructurado o **teorema de la estructura.**



1.1 | Teorema de la estructura

Según el teorema de la estructura, todo programa, por muy complejo que sea, puede ser diseñado con sólo tres tipos de instrucciones básicas:

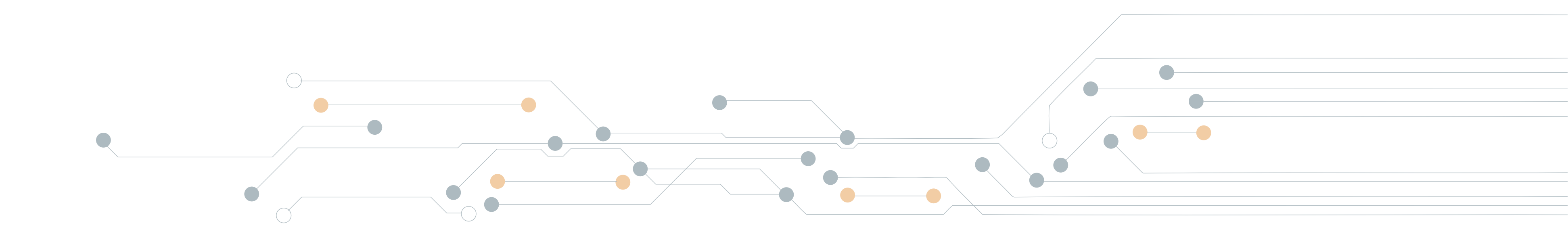
- Instrucciones secuenciales.
- Instrucciones alternativas o condicionales.
- Instrucciones repetitivas.

Este teorema, deja fuera de este grupo a instrucciones del tipo salto incondicional, que dificultan el seguimiento de los programas y los hace poco fiables.

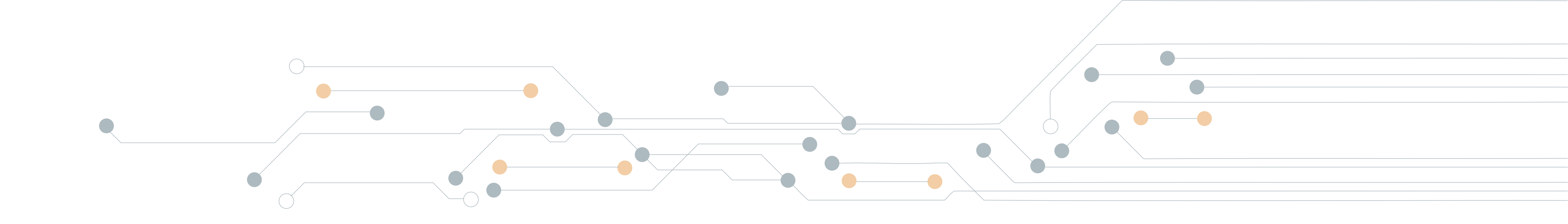
1.2 | Ventajas de la programación estructurada

Con la programación estructurada, diseñar algoritmos para resolver un problema informático sigue siendo una labor que requiere esfuerzo, creatividad y precaución, pero aplicando sus principios podemos obtener las siguientes ventajas:

- Programas más fáciles de comprender. Al carecer de saltos a otras partes del código, el algoritmo se puede seguir secuencialmente, lo que facilita su comprensión.
- Estructura de programa clara. Las tres tipos de instrucciones que se utilizan tienen un funcionamiento claro y definido.



- Reducción de errores. Al limitar el juego de instrucciones a utilizar y tener una estructura definida, se cometen menos errores al diseñar el algoritmo y por tanto, al programar.
- Facilidad en la detección de errores. No solamente se comenten menos errores, sino que los que se cometen son más fáciles de detectar y corregir.
- Programas más simples y rápidos. Estructurar un programa garantiza la solución más sencilla en cuanto a cantidad de código utilizado, además de más eficiente.



1.3 | Tipos de instrucciones

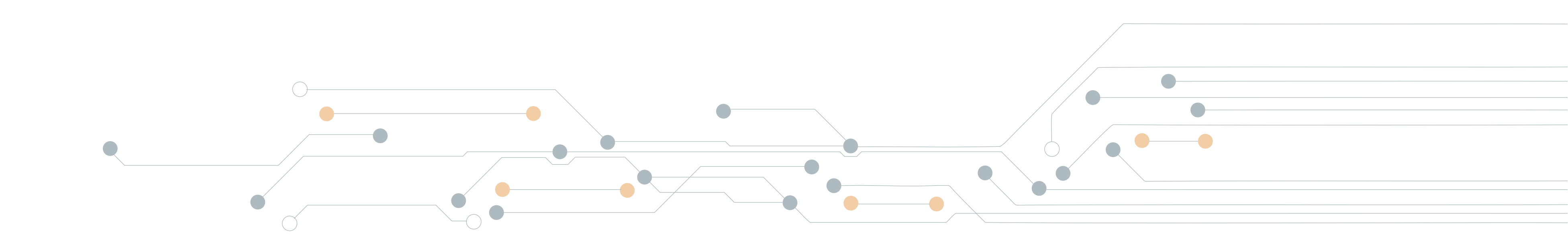
A continuación, vamos a ir analizado los tres tipos de estructuras lógicas que se utilizan en programación estructurada, para aplicarlos después en diversos ejercicios de ejemplo que iremos realizando.

INSTRUCCIONES SECUENCIALES

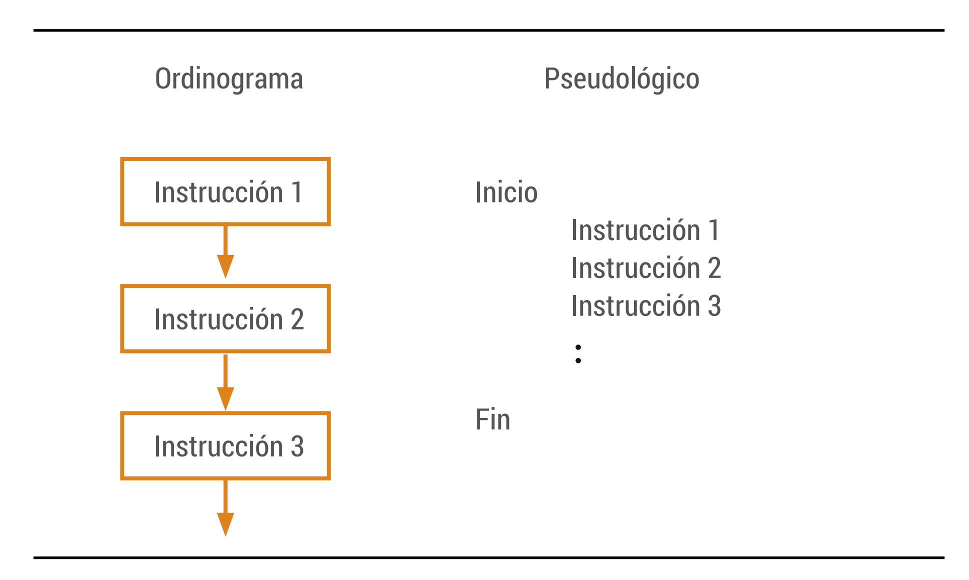
Instrucciones secuenciales son aquellas que se ejecutan en secuencia, es decir, una detrás de otra. Realmente, en un programa estructurado todas las sentencias son secuenciales, solo que unas pueden ser simples, como la inicialización de variables, operaciones aritméticas entre datos, lectura de números, y otras pueden ser complejas, como una instrucción alternativa o repetitiva.

Centrándonos en las operaciones simples, como hemos indicado, pueden consistir en la inicialización de alguna variable:

```
I=0
Operaciones entre datos:
I=I+4
A=B*I
O operaciones de entrada y salida:
Leer N
Imprimir Suma
```



En la siguiente imagen se ilustra cómo se representan estas instrucciones, tanto en un ordinograma como mediante pseudocódigo:

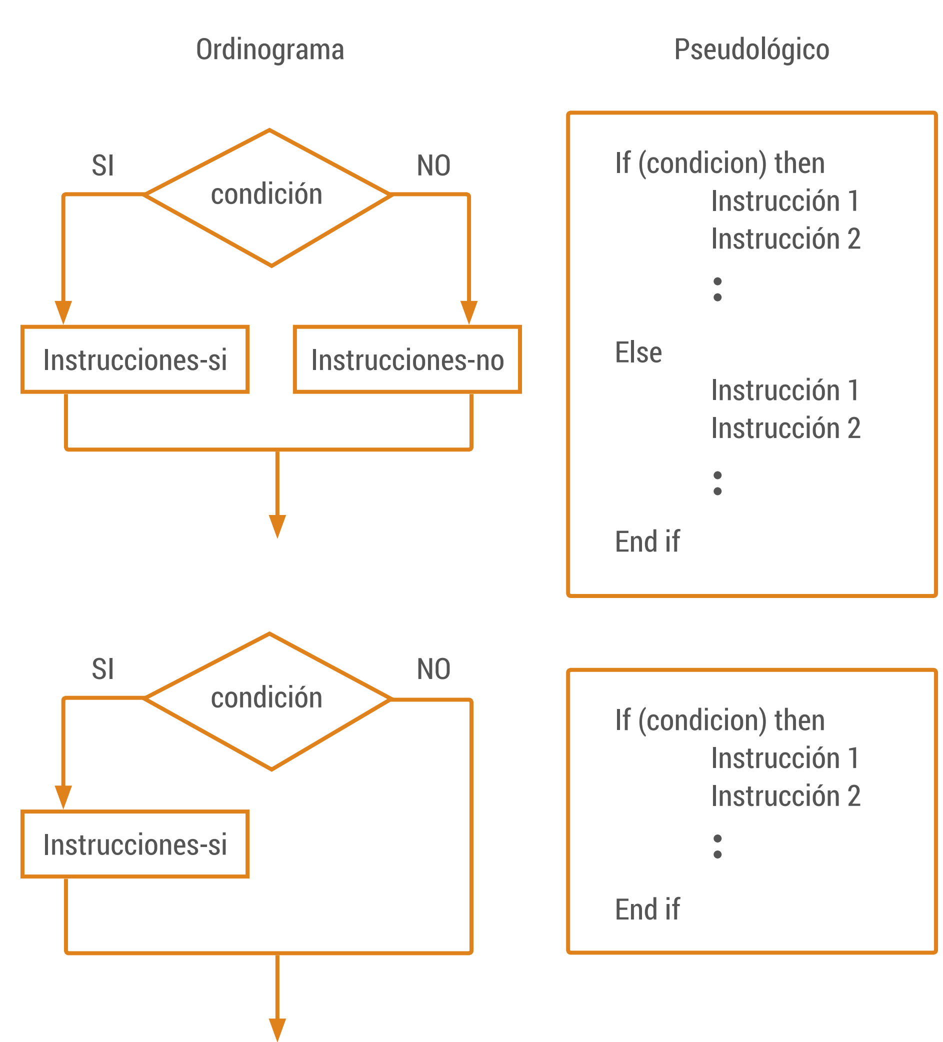


INSTRUCCIONES ALTERNATIVAS

Las instrucciones alternativas o condicionales nos permiten tomar decisiones en un programa, de modo que ante una condición el programa pueda tomar un camino en caso de que dicha condición se cumpla, o ejecutar sentencias diferentes si no se cumple.

Las instrucciones alternativas pueden ser de dos tipos:

- **Alternativas simples.** En este tipo de alternativas tenemos solamente dos caminos posibles, el que se tomará si la condición evaluada es verdadera y el que se seguirá si es falsa. En la siguiente imagen tenemos su representación tanto en ordinograma como en pseudocódigo:



Como puedes observar, hay dos variantes de esta instrucción, en la primera se evalúa la condición, si es verdadera se ejecutan unas instrucciones y si es falsa otras diferentes. Al final, el programa continúa por el mismo camino.

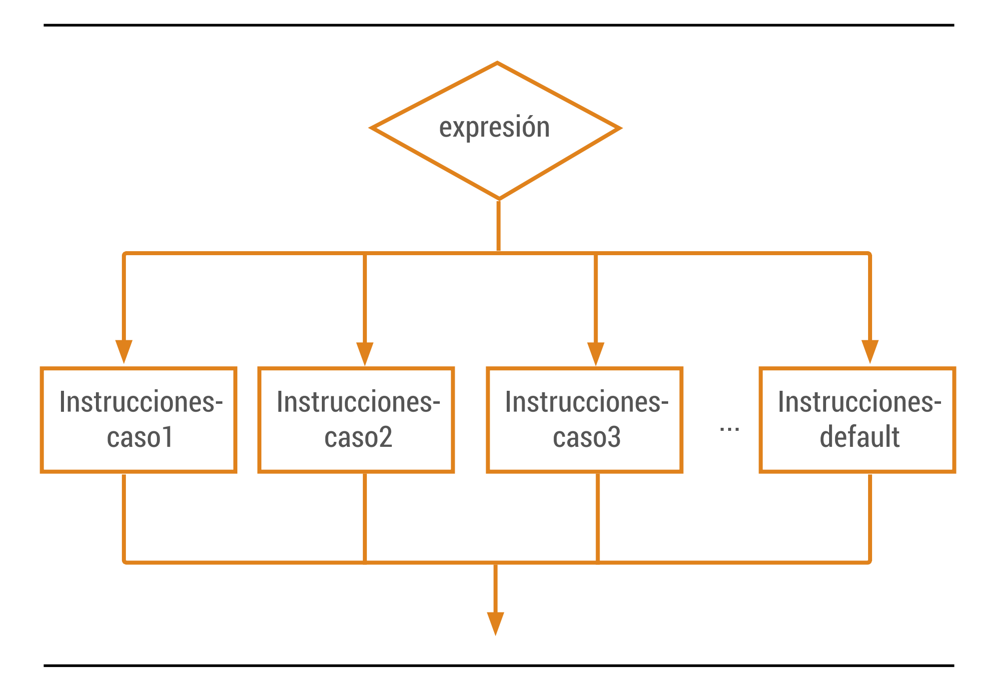
En la otra variante, si la condición es falsa no se ejecuta ninguna instrucción especial, el programa continúa por el mismo camino por el que seguiría después de ejecutar las instrucciones para la condición verdadera.

A la hora de expresar estas instrucciones mediante pseudocódigo utilizaremos el convenido de emplear palabras en inglés, de esta manera, puesto que las instrucciones que utilizan los lenguajes de programación de alto nivel para definir este tipo de estructuras son también palabras en inglés, se facilita la traducción final del diseño en pseudocódigo a código en la fase de implementación.

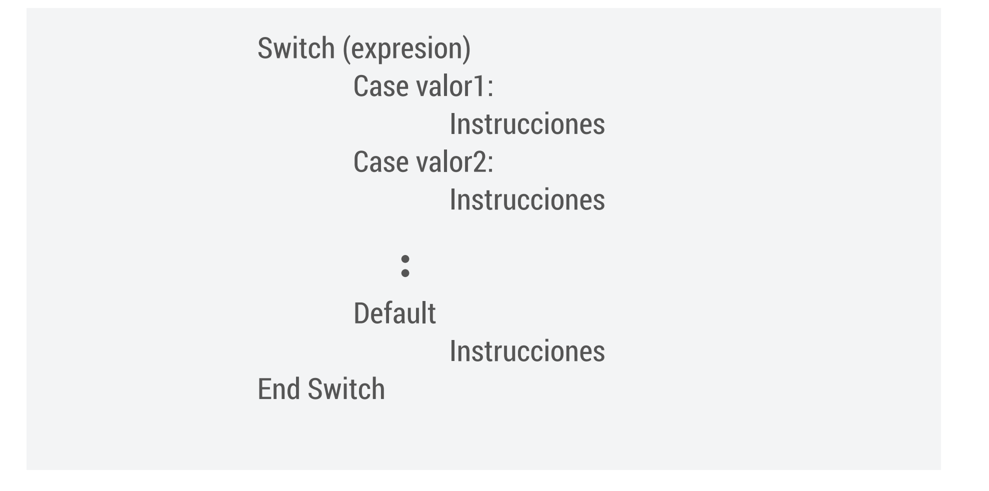
- **Alternativas múltiples.** En una estructura de tipo alternativa múltiple, no es una condición de tipo verdadero o falso la que determina el camino a seguir por el programa, sino una expresión que puede dar como resultado múltiples valores.

Para cada valor que nos interese controlar se definirá un grupo de instrucciones a ejecutar diferente. Opcionalmente, se puede definir un conjunto de instrucciones a ejecutar en caso de que el resultado de la expresión no coincida con ninguno de los valores controlados. Tras la ejecución de cualquiera de los casos, el programa continuará por un camino común.

A continuación se muestra el diagrama de flujo equivalente a esta instrucción:



En pseudocódigo, podríamos definirlo de la siguiente manera:



```
Switch (expresion)
        Case valor1:
                Instrucciones
        Case valor2:
                Instrucciones
            :
        Default
                Instrucciones
End Switch
```

A la hora de indicar los valores de cada caso, podemos utilizar un valor concreto, una lista de valores separados por comas o un rango de valores definido por la expresión *valorinicial To valorfinal*, si bien algunos lenguajes de programación de alto nivel solo permiten indicar valores concretos a la hora de evaluar las salidas de una instrucción de este tipo.

El final de las instrucciones de cada caso vendrá determinado por el caso siguiente, y tras ejecutar un caso, el programa continuará con la ejecución de la sentencia indicada después del *End Switch*

En el siguiente ejemplo te mostramos un pseudocódigo de un algoritmo encargado de leer una nota de un examen y mostrar la calificación correspondiente:

```
Inicio
        Datos:
        nota entero
        Algoritmo:
        Leer nota
        Switch(nota)
                Case 0 to 4
                        Imprimir “Suspenso”
                Case 5
                        Imprimir “Aprobado”
                Case 6
                        Imprimir “Bien”
                Case 7, 8
                        Imprimir “Notable”
                Case 9, 10
                        Imprimir “Sobresaliente”
                Default
                        Imprimir “Nota no válida”
        End Switch
Fin
```

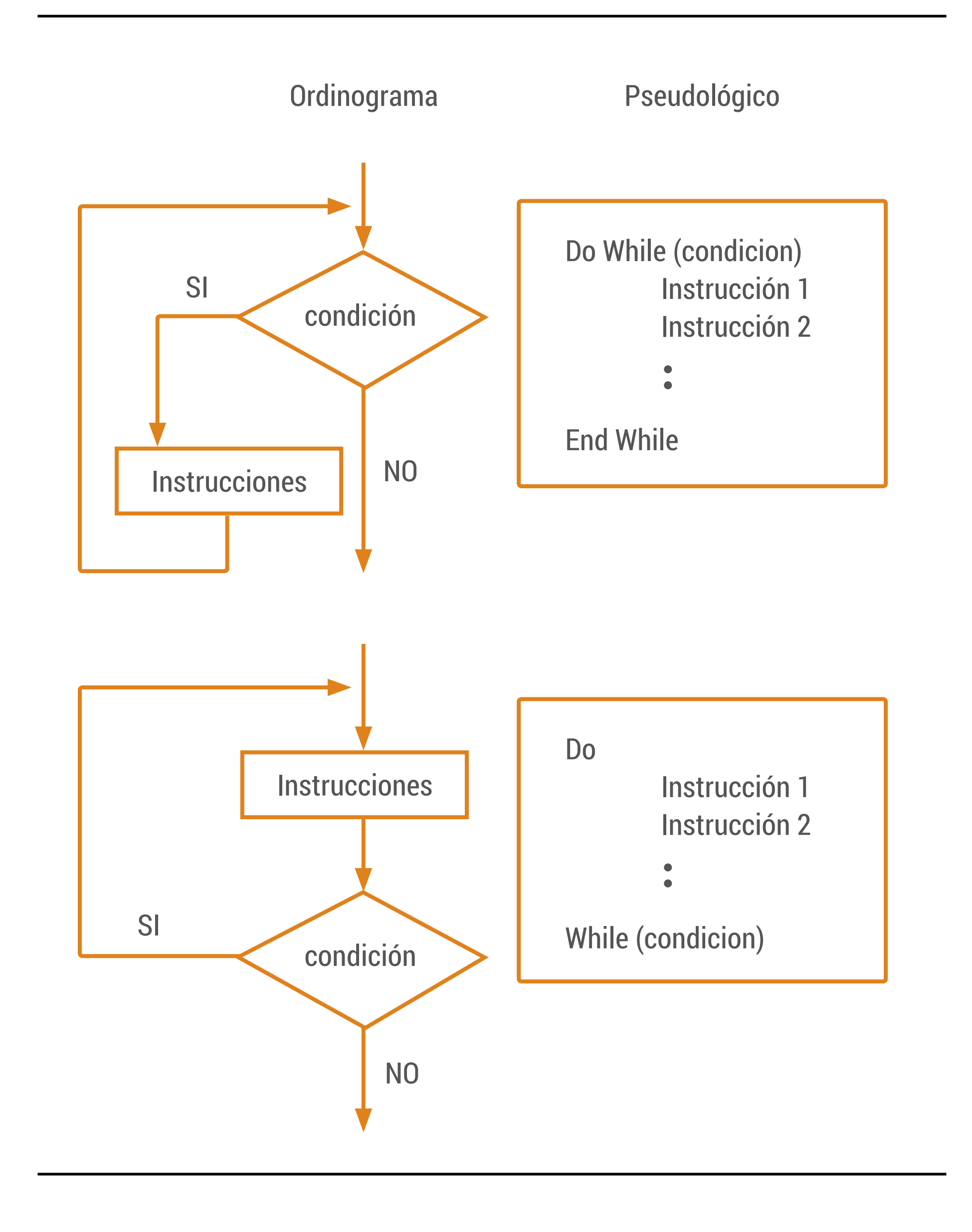


INSTRUCCIONES REPETITIVAS

Las instrucciones de tipo repetitivo permiten ejecutar un bloque de sentencias varias veces en función del cumplimiento de una condición. Se trata de un tipo de instrucción muy importante en programación, con la que podemos realizar cálculos iterativos y recorrer tablas u otras estructuras de datos complejas.

Utilizando adecuadamente esta instrucción, podemos evitar saltos en un programa, que es una de las claves de la programación estructurada.

En el siguiente esquema tenemos las dos formas clásicas en las que se puede presentar este tipo de instrucciones:



En el primer caso, lo primero es evaluar la condición y si esta se cumple, se ejecutará el bloque de sentencias definido entre **Do While** y **End While**. Al finalizar la ejecución de dicho bloque, se vuelve a evaluar la condición de nuevo y si vuelve a cumplirse, otra vez se ejecuta el bloque de sentencias. Es de esperar que la ejecución del bloque de sentencias actúen sobre los elementos utilizados en la condición, de forma que esta deje de cumplirse en algún momento.

En el segundo caso, primero se ejecutan las instrucciones indicadas dentro del **Do.. While** y después se evalúa la condición, de modo que si esta se cumple, el bloque de instrucciones volverá a ejecutarse de nuevo. Con esta estructura se garantiza que el bloque de instrucciones se ejecuta al menos una vez, aunque la condición no se cumpla desde el principio.

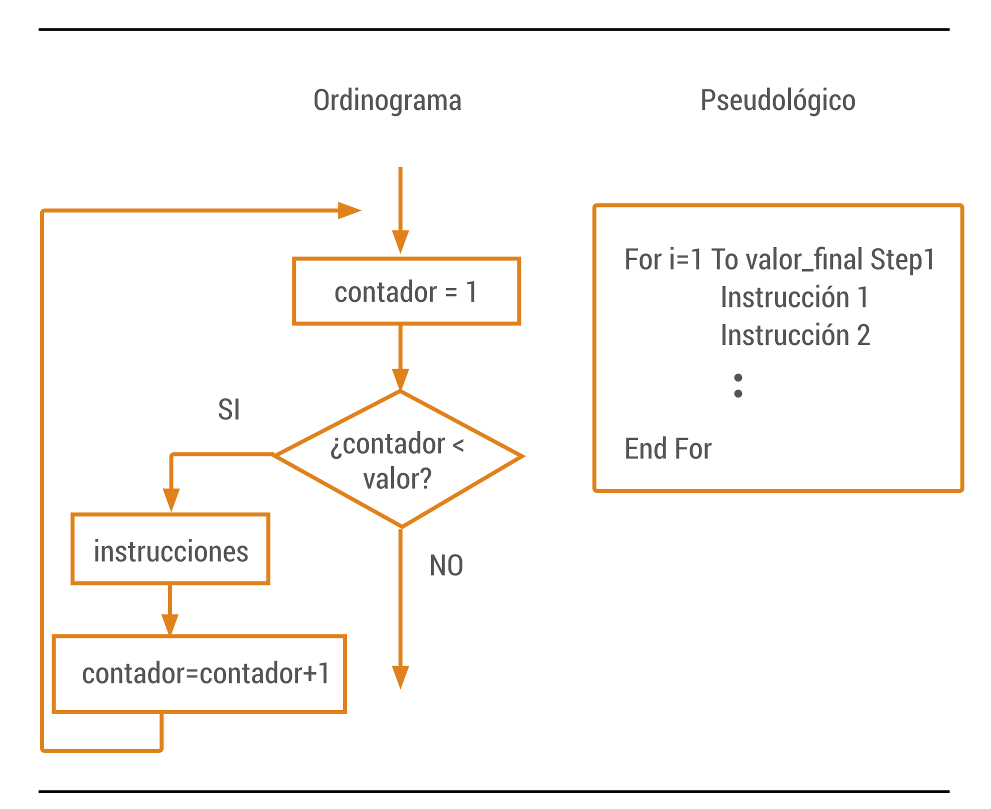
Podemos ver un ejemplo de utilización de esta segunda forma de la estructura repetitiva While en la siguiente versión del programa de las notas en la que se repite la lectura de la nota hasta que el valor introducido sea válido, es decir, entre 0 y 10:

```
Inicio
      Datos:
      nota entero
      Algoritmo:
Do
            Leer nota
      While(nota<0 OR nota>10)
      Switch(nota)
            Case 0 To 4
                  Imprimir “Suspenso”
            Case 5
                  Imprimir “Aprobado”
            Case 6
                  Imprimir “Bien”
            Case 7, 8
                  Imprimir “Notable”
            Case 9, 10
                  Imprimir “Sobresaliente”
      End Switch
Fin
```



Un caso habitual de instrucción repetitiva es aquella que ejecuta el bloque de instrucciones un número concreto de veces, el cual viene determinado por una variable contador que se va incrementando en cada iteración. Este tipo de instrucciones repetitivas suele representarse en los lenguajes de programación de alto nivel mediante una instrucción llamada **For** en vez de *While.*

La siguiente ilustración nos muestra su representación en forma de ordinograma y pseudocódigo:



Desde que la variable de control, llamada i en este caso, toma un valor inicial hasta que alcanza un valor final, se ejecutan las instrucciones del bloque. Al entrar en la instrucción *for*, la variable se inicializa al valor indicado, en este caso 1 y, si el valor de esta variable es menor o igual que el valor final, se ejecuta el bloque de sentencias definido en su interior. Al finalizar la ejecución del bloque, se incrementa en 1 la variable de control y se vuelve a comprobar si ha alcanzado el valor final, así hasta que la variable haya recorrido todos los valores enteros entre el inicial y el final.

En la instrucción *step* se indica el incremento a realizar al finalizar cada iteración y, aunque normalmente es 1, puede ser cualquier otro valor.

El algoritmo del programa encargado de calcular la suma de todos los números naturales comprendidos entre 1 y un número leído podría representarse de la siguiente manera:

```
Inicio
      Datos:
      N entero
      I entero
      Suma entero
      Código:
      Suma =0
      Leer N
      For I=1 to N Step 1
            Suma=Suma+I
      End For
      Imprimir “La suma final es “,Suma
Fin
```

2. Otros elementos de programación estructurada

Como ya hemos visto, un algoritmo es un conjunto de operaciones que resuelven un problema específico. Pero muchos problemas de programación, especialmente los que tienen cierta complejidad, pueden resolverse con diferentes algoritmos, algunos de ellos más óptimos y eficientes que otros.

2.1 | Tipos de datos

En todos los programas se manejan datos que, como ya hemos visto en los pseudocódigos realizados, se almacenan en variables y se opera con ellos a través de estas.

Cada lenguaje de programación de alto nivel tiene su propio juego de tipos de datos, pero de cara a poder definir los distintos tipos de datos con los que vamos a tratar en un pseudocódigo, vamos a definir nuestro propio conjunto de tipos. Será un conjunto básico, formado por los tipos más comunes:

Nombre tipo	Significado
Integer	Numérico entero
Decimal	Numérico decimal
String	Texto
Date	Fecha
Boolean	Lógico (true o false)

Así, en la zona de definición de datos, cuando vayamos a indicar el tipo de una variable utilizaremos los nombres indicados en la tabla anterior.

2.2 | Operadores

Los operadores permiten realizar operaciones con los datos dentro de un programa. Ya hemos visto como utilizar algunos de los más habituales en los ejemplos que te hemos ido presentando a lo largo del módulo.

En la siguiente tabla resumimos los operadores que utilizaremos en el diseño de nuestros programas, clasificados por tipos:

Tipos de operadores	Operadores
Aritméticos	+, -, *, /, %
Condicionales	<, >, <=, >=, = =, <>
Lógicos	AND, OR, NOT

Los operadores aritméticos los utilizaremos con tipos numéricos, es decir, Integer y Decimal. Los cuatro primeros son para las operaciones de suma, resta, multiplicación y división, mientras que el operador "%" se utiliza para calcular el resto de la división entre dos números. Este operador ya lo utilizamos en los ejercicios donde teníamos que comprobar si un número era par o impar.

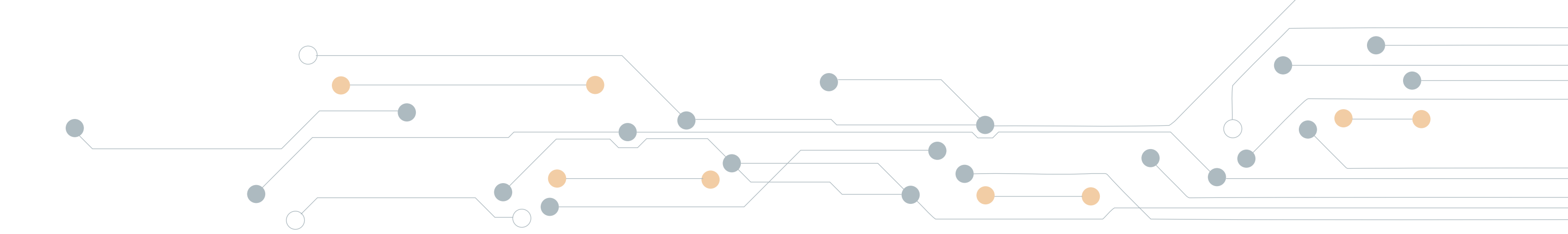
Los operadores condicionales se emplean en las condiciones de las instrucciones de tipo If y While. Siempre devolverán un valor de tipo boolean (true o false). Los operadores <, >, <=, >= los emplearemos con tipos numéricos, mientras que los de igualdad y desigualdad (== y <>) los emplearemos con cualquier tipo de datos

En cuanto a los operadores lógicos, se usarán para realizar las tres operaciones lógicas básicas:

- **AND.** El resultado es verdadero si las dos expresiones lo son:

```
(3>1 AND 5<10)  // el resultado es vedadero

(4>2 AND 2>8)  //el resultado es falso porque la segunda expresión // lo es
```



- **OR.** El resultado es verdadero si alguna de de las expresiones lo es:

```
(10>3 OR 5<1)  //el resultado es verdadero
```

- **NOT.** El contrario a un valor boolean:

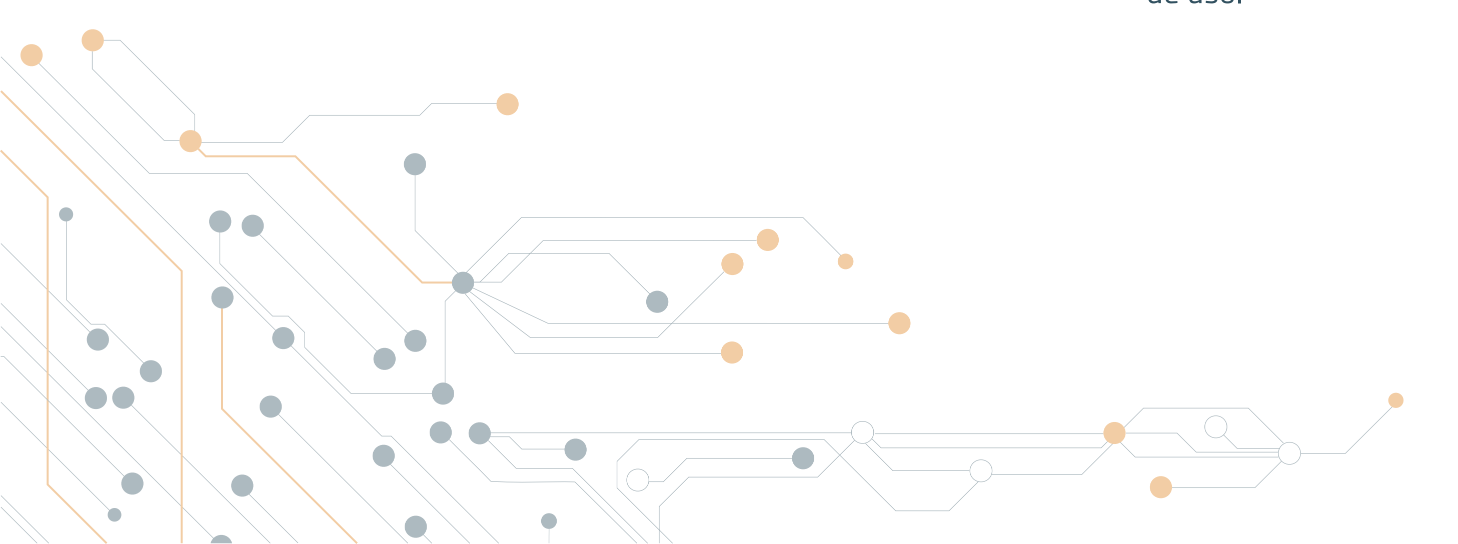
```
(NOT 4<6) //el resultado es falso
```

2.3 | Comentarios

Los comentarios los utilizamos en un programa, y también en pseudocódigo, para incluir notas del programador que no forman parte de la sintaxis lógica del programa.

Su misión es aclarar a la persona que está leyendo el programa el uso de alguna instrucción o expresión, por lo que ayudan a entender el funcionamiento del mismo.

Los comentarios los podemos incluir en cualquier parte del pseudocódigo, y siempre irán precedidos por los símbolos //. En la explicación de los operadores lógicos hemos visto unos ejemplos de uso.




Telefónica
EDUCACIÓN DIGITAL